



Modernizing your application by migrating from an RDBMS to Amazon
DynamoDB

AWS Prescriptive Guidance



AWS Prescriptive Guidance: Modernizing your application by migrating from an RDBMS to Amazon DynamoDB

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Introduction	1
DynamoDB overview	2
Features and benefits	2
Partition keys	3
Indexes	3
Time to Live	4
Pricing models	4
Transactions	6
Large components	7
Backup and restore	7
Language and SDK support	8
Sample application	9
Old data access pattern	9
New data access pattern	10
RDBMS table schema and mapping	11
Single table design	13
Global secondary indexes	14
Access patterns	15
Object persistence interface	15
Document interface	16
Low-level API	17
Converters	18
Best practices	20
Use the object persistence access pattern	20
Choose the correct capacity provisioning mode	20
Use caching	20
Use queries instead of scans	20
Validate data integrity	21
FAQ	22
What is the maximum table size that I can create in DynamoDB?	22
How many tables can I create per account?	22
How many global secondary indexes can I create on a DynamoDB table?	22
How many items can I add or modify per transaction?	22
Next steps and resources	23

Document history	24
Glossary	25
#	25
A	26
B	29
C	30
D	33
E	37
F	39
G	40
H	41
I	42
L	44
M	45
O	49
P	51
Q	53
R	53
S	56
T	59
U	61
V	61
W	62
Z	63

Modernizing your application by migrating from an RDBMS to Amazon DynamoDB

Ramkumar Ramanujam and Mahesh Kumar Vemula, Amazon Web Services (AWS)

February 2023 ([document history](#))

As an organization expands its business, the load on its information systems increases significantly. Performance optimization measures can help address this increased demand only to some extent. Often times, the increased demand leads to a point where the database isn't able to handle the load. This issue is especially prevalent in applications that are built on a relational database management system (RDBMS). Although major RDBMS providers offer ways to handle system load through read replicas, database mirroring, primary-secondary configurations, and similar features, the cost and licensing requirements for these features might become a concern. For organizations that are interested in researching alternatives for data storage, this guide focuses on migrating from an RDBMS to Amazon DynamoDB.

The first part of this guide provides an overview of DynamoDB features and benefits. The second part of the guide is based on a case study of an application that was migrated from an RDBMS (Microsoft SQL Server) to DynamoDB. It provides code examples to address two migration challenges: mapping relational data to the DynamoDB document structure and key-value collection, and changing the data access layer of the application to support create, read, update, and delete (CRUD) operations in DynamoDB.

This guide is for program or project managers, database administrators, and database architects who are planning to migrate their RDBMS systems to DynamoDB to address their rapid application development (RAD) and high performance requirements. The guide assumes a basic understanding of relational databases and NoSQL concepts, but doesn't require DynamoDB skills or experience.

DynamoDB overview

Amazon DynamoDB is a NoSQL key-value and document database that provides fast and predictable performance with seamless scalability. It's a fully managed, multi-Region, multi-active, and durable database. (The global table feature in DynamoDB synchronizes modifications made in one AWS Region to all other selected Regions automatically, to provide multi-active support.) DynamoDB has built-in security capabilities, backup and restore options, and in-memory caching for internet-scale applications.

The schemaless nature of NoSQL databases helps reduce the time and process required for pushing a database (read schema) change to production, thereby enabling rapid application development (RAD). NoSQL databases such as DynamoDB are built for high-performance read/write operations.

Features and benefits

DynamoDB provides the following features and benefits:

- No servers to manage – DynamoDB is a fully managed NoSQL database service, which means that it doesn't involve any server maintenance overhead.
- Schemaless – DynamoDB supports rapid application development and deployments.
- Performance at scale – DynamoDB provides fast and predictable performance with seamless scalability.
- ACID support – DynamoDB supports atomicity, consistency, isolation, and durability (ACID) transactions, to help you maintain data correctness.
- Highly availability and durability – Your data is stored on solid-state disks (SSDs) and is automatically replicated across multiple Availability Zones in an AWS Region, providing built-in high availability and data durability.
- Automatic scaling – DynamoDB uses the AWS Application Auto Scaling service to dynamically adjust provisioned throughput capacity in response to traffic patterns.
- Flexible pricing options – DynamoDB provides two capacity modes with specific billing options: on-demand capacity mode and provisioned capacity mode.
- Point-in-time recovery – You can enable continuous backups using point-in-time recovery to help protect your tables from accidental write or delete operations, and restore your tables to any point in time during the last 35 days.

- Time to Live (TTL) – You can automatically delete items from a DynamoDB table after a specified duration.
- Global tables – You can deploy multiple replicas across AWS Regions without having to build your own replication solution.
- Global secondary indexes – You can query your DynamoDB table by using partition and sort keys that are different from the table's own partition and sort keys.
- DAX – The DynamoDB Accelerator (DAX) caching service provides sub-millisecond response times for read operations.
- DynamoDB Streams – This feature provides a time-ordered sequence of item-level modifications in a log to support real-time tracking and notification of changes in DynamoDB tables.

For more information about these benefits, see [Amazon DynamoDB Features](#) on the AWS website. The following sections discuss some of the features that are relevant to modernizing your database workloads by migrating from an RDBMS to DynamoDB.

Partition keys

DynamoDB is schemaless, so it's not necessary to define all attributes for a table. The partition key attribute is required, and the sort key is optional. The rest of the attributes are arbitrary and can vary per item. We recommend that you choose a partition key with high cardinality so that frequently accessed items don't reside on the same partition. This practice helps you avoid data access imbalances and hot partitions. For more information, see [Best practices for designing and using partition keys effectively](#) in the DynamoDB documentation.

Indexes

Indexes give you access to alternate query patterns and can speed up queries. You should create indexes carefully, regardless of whether you're using a relational database or DynamoDB. Whenever a write operation occurs on a table, all the indexes for that table must be updated.

A global secondary index contains a selection of attributes from the base table, but they are organized by a primary key that is different from the table's own primary key. In DynamoDB, global secondary indexes are sparse by default. That is, the sort key is optional and doesn't appear in every table item. To take advantage of this feature, you can create global secondary indexes that store and project only the required attributes. You can have up to 20 global secondary indexes on a DynamoDB table. For more information about this feature, see [Using global secondary indexes in DynamoDB](#) in the DynamoDB documentation.

Time to Live

You can set a Time to Live (TTL) property on a DynamoDB table to define a per-item (record) timestamp to specify when an item is no longer needed. Shortly after the specified timestamp, DynamoDB deletes the item from the table without consuming any additional capacity units. For more information about this feature, see [Expiring items by using DynamoDB Time to Live](#) in the DynamoDB documentation.

Pricing models

DynamoDB offers two pricing models to choose from: provisioned capacity and on-demand capacity. Your choice of the pricing model depends on your projected workloads.

Pricing model	Workload type	Cost	Read/write throughput
Provisioned capacity	Predictable	Lower	<p>You specify the number of read/write operations per second in terms of read capacity units (RCUs) and write capacity units (WCUs). For example:</p> <ul style="list-style-type: none">• For an item up to 4 KB in size, 1 RCU can perform 2 eventually consistent reads per second.• For an item up to 1 KB in size, 1 WCU can perform 2 eventually consistent reads per second.

Pricing model	Workload type	Cost	Read/write throughput
			<p>You can enable automatic scaling to adjust capacity in response to traffic changes.</p>
On-demand capacity	Dynamic	Higher	<p>You don't specify throughput requirements. DynamoDB accommodates your workloads automatically.</p> <p>You're charged for the reads and writes that your application performs on your tables in terms of read request units and write request units. For example:</p> <ul style="list-style-type: none"> An 8 KB item requires 1 read request unit for an eventually consistent read and 8 write request units for a non-transactional write operation.

For more information about these two models, see [Read/write capacity mode](#) in the DynamoDB documentation.

Transactions

DynamoDB supports atomicity, consistency, isolation, and durability (ACID) transactions across one or more tables within a single AWS account and AWS Region.

To manage changes to multiple items within and across tables, you can use the DynamoDB transactional `TransactWriteItems` and `TransactGetItems` APIs.

- `TransactWriteItems` is a batch operation that contains a write set with one or more `PutItem`, `UpdateItem`, and `DeleteItem` actions. `TransactWriteItems` can optionally check for prerequisite conditions that must be satisfied before making updates. These conditions might involve the same items as those in the write set, or different items. If any condition isn't met, the transaction is rejected.
- `TransactGetItems` is a batch operation that contains a read set with one or more `GetItem` actions. If you issue a `TransactGetItems` request on an item that is part of an active write transaction, the read transaction is canceled. To get the previously committed value, you can use a standard read operation.

For more information about these APIs, see [Amazon DynamoDB transactions: how it works](#) in the DynamoDB documentation.

Limitations

DynamoDB transactional API operations have the following constraints:

- A transaction cannot update more than 100 unique items.
- A transaction cannot contain more than 4 MB of data.
- No two actions in a transaction can work against the same item in the same table. For example, you cannot run both `ConditionCheck` and `Update` actions on the same item in one transaction.
- A transaction cannot operate on tables in more than one AWS account or Region.
- Transactional operations provide ACID guarantees only within the AWS Region where the write operation originally takes place. Transactions are not supported across Regions in global tables.

- The object persistence model doesn't support transactions. To use the transaction feature, you must access the database and tables using the [DynamoDB low-level API](#).

Large components

DynamoDB has a size limit of 400 KB for each item. This limit includes both attribute name (binary length with UTF-8 encoding) and attribute value (again binary length). The attribute name counts toward the size limit. For example, consider an item that has two attributes: one attribute named "country-code" with value "IN" and another attribute named "country-phone-prefix" with value "91". The total size of that item is 36 bytes.

Workaround

If an item is associated with many attributes and properties, or a large amount of data, its size might exceed 400 KB. In this case, you can store the serialized item in Amazon Simple Storage Service (Amazon S3) in JSON format, and save the Amazon S3 location as an attribute (`S3Location`) in the item. Read and write operations for that item fetch the S3 object and update the JSON string. The primary key, sort key, and all attributes used by local indexes and global secondary indexes should be stored in the table along with the `S3Location` attribute. This requires additional logic in the application (data access layer) to check for the `S3Location` attribute and to fetch the complete item data from Amazon S3.

Backup and restore

Backup and restore support is a common feature expectation in any database. DynamoDB natively supports backup and restore operations within the same account, but you can perform table copy across multiple accounts by using other options or processes. These processes do not consume any read/write capacity units. For more information, see the guide [Cross-account full table copy options for Amazon DynamoDB](#) on the AWS Prescriptive Guidance website.

Limitations

DynamoDB currently supports cross-account backup and restore by using [AWS Backup](#), but the account has to be part of the same organization. You can address this limitation by adopting one of the following solutions:

- Custom implementation in a programming language of your choice (for example, .NET, Java, or Python) by using an [AWS SDK](#). You can scan items from the source table in account A and

write items (`BatchWrite`) to a table in account B. You can run this code on the server, on an on-premises computer, or in AWS Lambda (if the database is small and the script takes less than 15 minutes to run). For more information, see the pattern [Copy Amazon DynamoDB tables across accounts using a custom implementation](#) on the AWS Prescriptive Guidance website.

- Using AWS Data Pipeline. You can export (scan) from the source table and import (write) into the target table. For more information, see [Exporting and importing DynamoDB data using AWS Data Pipeline](#) in the DynamoDB documentation.
- Using AWS Glue. For information about this option, see the guide [Cross-account full table copy options for Amazon DynamoDB](#) on the AWS Prescriptive Guidance website.

Language and SDK support

The [AWS SDKs](#) provide a simplified programming interface to AWS services and support for .NET, Java, JavaScript, Node.js, Python, PHP, and Ruby.

You can choose from three patterns to access DynamoDB tables by using an AWS SDK: object persistence model (high-level interface), document interfaces, and low-level interfaces. For detailed information, see [Access patterns](#) later in this guide.

Sample application

This section provides guidance for teams that are evaluating a migration from their relational database management system (RDBMS) to a NoSQL database, and focuses on Amazon DynamoDB as the target NoSQL database. It addresses the following two challenges, based on a case study of an application that migrated from Microsoft SQL Server to DynamoDB:

- Mapping relational data from multiple tables in the RDBMS to a document structure and key-value collection in DynamoDB
- Changing the data access layer in the application to perform create, read, update, and delete (CRUD) operations in DynamoDB

The discussion and guidance includes code examples written in C#, using the AWS SDK for .NET.

The sample web application maintains the configuration for hundreds of applications used in an organization, including allowed users and hosts (web, mobile, desktop) for each application, metadata, search keywords, and so on. The application provides configuration maintenance and search functionality for different versions of various applications used in the organization. Configuration changes are tracked by using audit tables. Here's a typical workflow for the sample application:

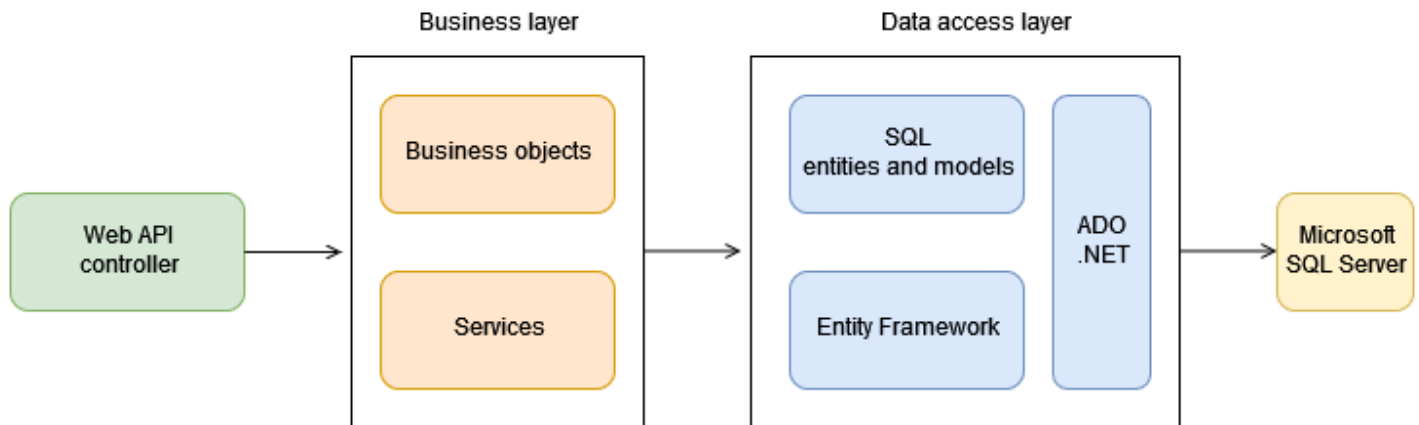
1. Create a configuration for the test application.
2. Promote the test application configuration to production (that is, create a production application configuration).
3. Update and audit changes (create an audit record, call the changed application configuration).

Old data access pattern

The source technology stack consisted of the following:

- ASP.NET Web API controller
- Business objects
- ASP.NET Entity Framework (EF)
- ADO.NET Data Services

- Microsoft SQL Server 2016

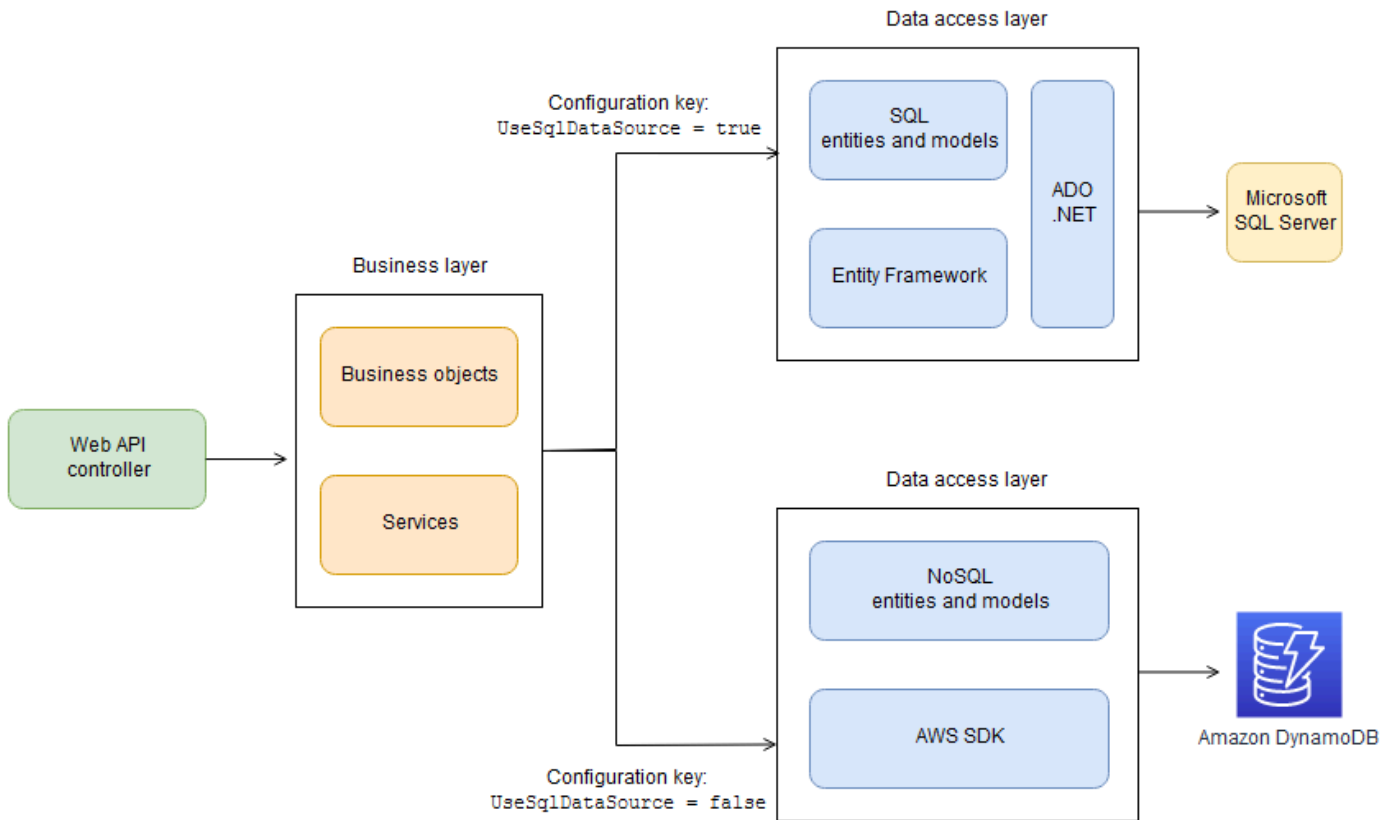


New data access pattern

The migrated application supports both SQL Server and DynamoDB based on the configuration key (`UseSqlDataSource`) provided in the configuration file. As shown in the following diagram, if the value of `UseSqlDataSource` is `true`, the application connects to SQL Server. If the value is `false`, the application connects to DynamoDB.

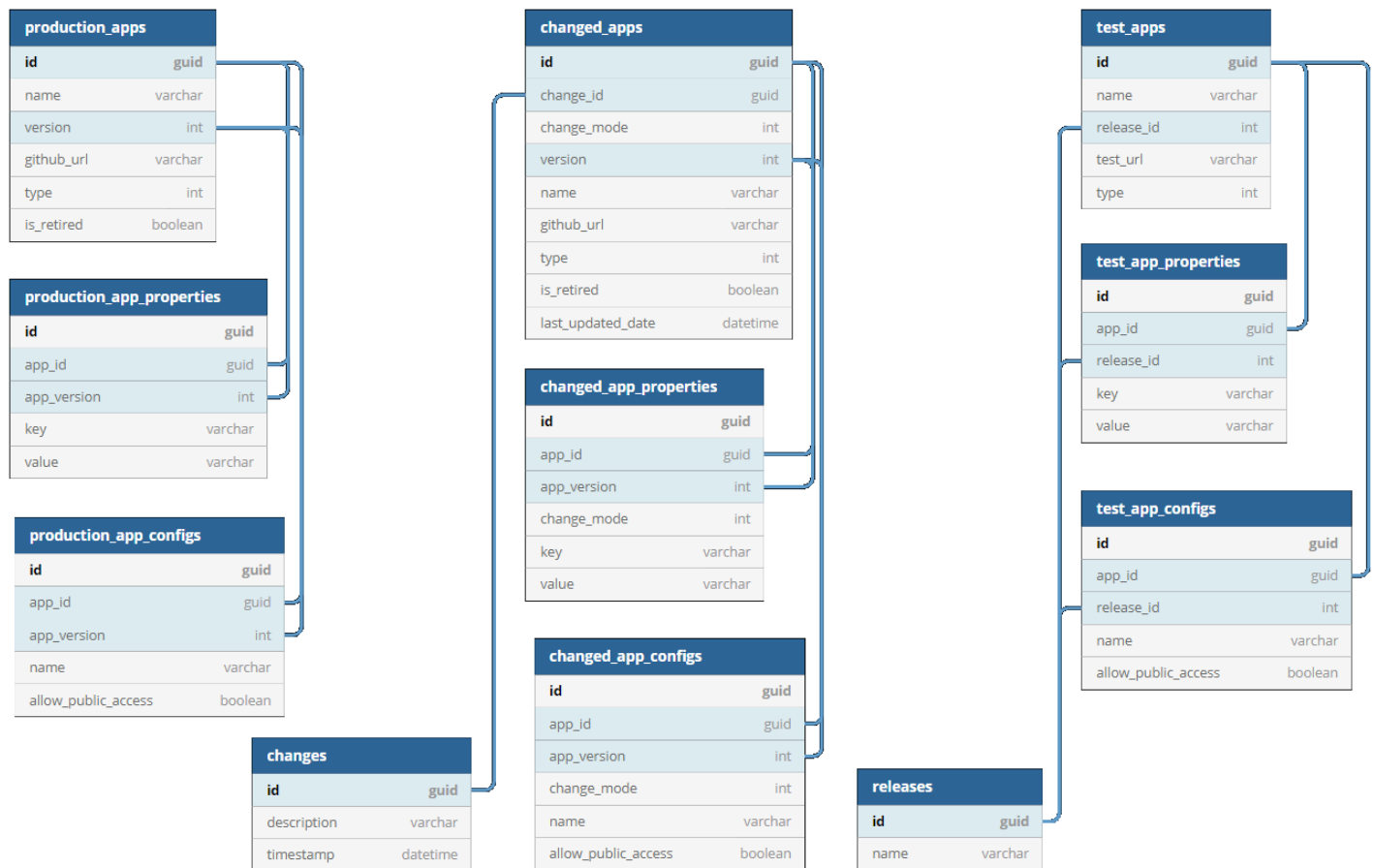
The new technology stack consists of the following:

- ASP.NET Web API controller – Accepts HTTP requests over various API endpoints.
- Business objects and services – Classes and objects that have the business logic to process input and data fetched from the database.
- NoSQL entities and models – Classes that map to items stored in DynamoDB.
- AWS SDK – Provides programmatic access to DynamoDB and other AWS services.
- DynamoDB – Database for storing application data.

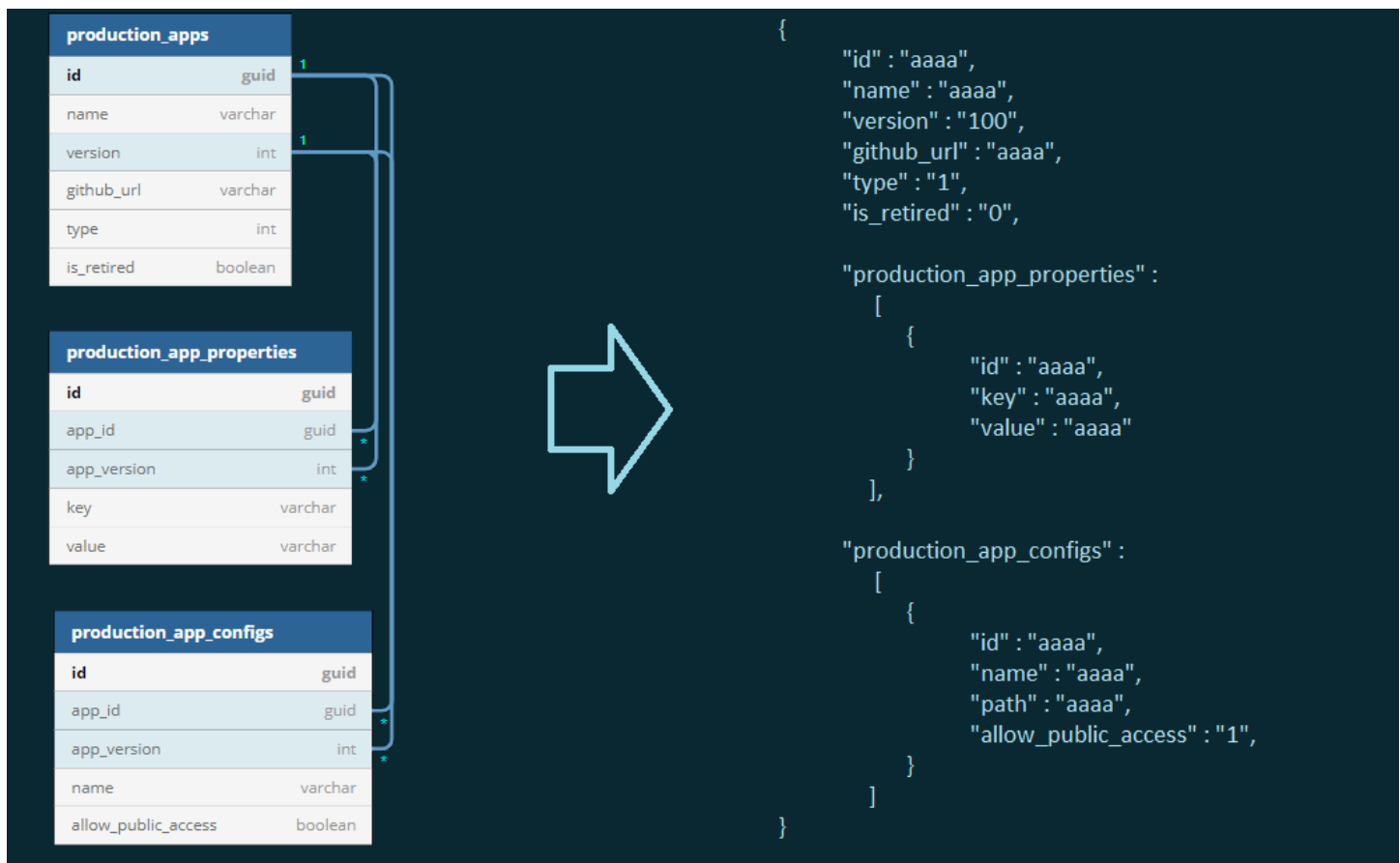


RDBMS table schema and mapping

The following diagram shows the tables and relationships in the source RDBMS schema.



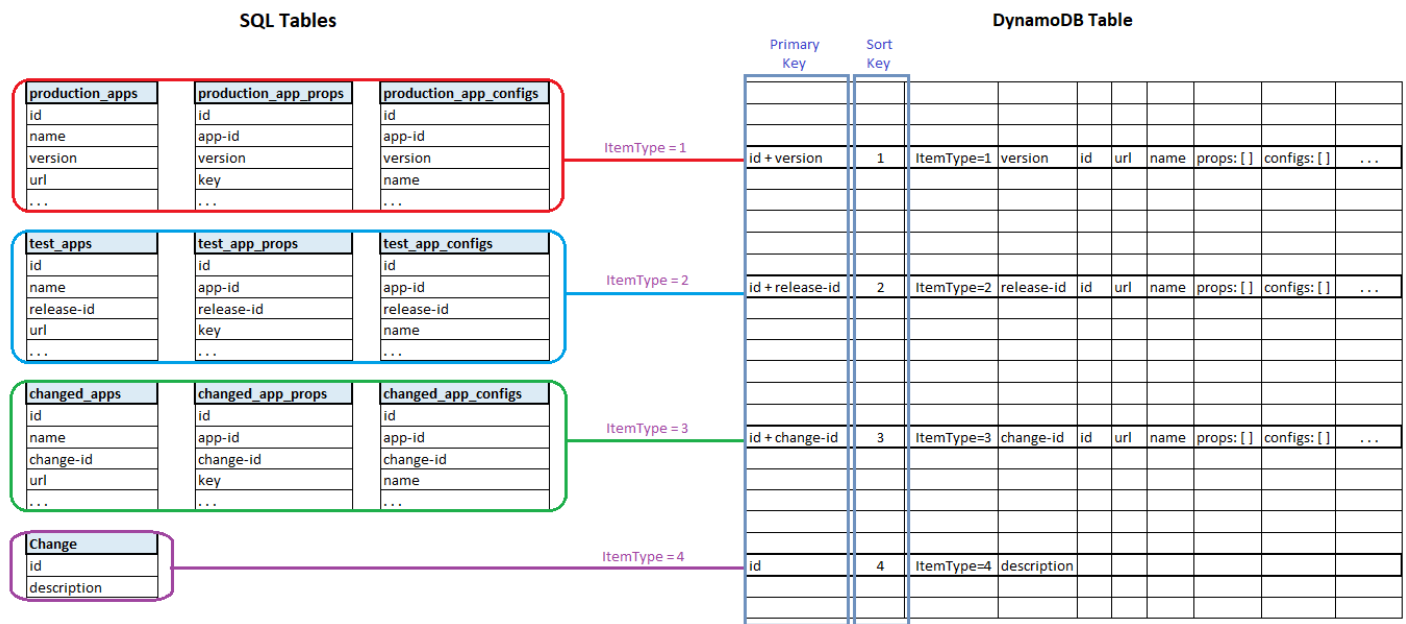
As shown in the previous diagram, the `production_apps` table has `id` and `version` columns, which have a one-to-many relationship with the `production_app_properties` and `production_app_configs` tables. Therefore, in the DynamoDB design, the `production_app_properties` and `production_app_configs` tables are embedded within the `production_app` item, as shown in the following JSON code. Because `production_app_properties` and `production_app_configs` can have multiple values, these tables are added as arrays in the JSON code. The `changed_apps` and `test_apps` tables are mapped similarly.



Single table design

DynamoDB doesn't maintain relationships; it supports a fixed table schema. As a result, you can store different types of items (such as SQL tables) in a single DynamoDB table, with an attribute (`ItemType`) that identifies the type of item.

In DynamoDB, the combination of partition keys (PK) and sort keys (SK) must be unique, so these keys vary based on the item type.



Global secondary indexes

Indexes help retrieve data faster and improve application performance. For the sample application, the following indexes were created. The PKs and SKs were chosen based on how distinct items could be identified.

Index name	Description	Partition key (PK)	Sort key (SK)	Projected Attributes
Version-index	Gets all production applications for a given version.	version		id, name
Release-index	Gets all test applications for a given release-id .	release-id		id, name

Index name	Description	Partition key (PK)	Sort key (SK)	Projected _Attributes
Change-in dex	Gets all (changed) applications associated with a change-id .	change-id		id, modified-by, date

Access patterns

As mentioned earlier in this guide, you can choose from three access patterns to perform create, read, update, and delete (CRUD) operations on DynamoDB tables: object persistence interface, document interfaces, and low-level API interface. The following sections describe each interface. For our SQL Server to DynamoDB use case, we chose the object persistence interface for simplicity, readability, and ease of maintenance.

Object persistence interface

The object persistence interface provides a high-level, abstracted access mechanism for performing CRUD operations on DynamoDB items by using .NET models, similar to Entity Framework entities. The interface properties map to DynamoDB item attributes. The AWS SDK for .NET supports custom property attributes in this model to customize the serialization and deserialization of individual properties, to handle null values, and for type conversions.

Sample model used in the application:

```
[DynamoDBTable("AppLibrary")]
public class ProdApp
{
    [DynamoDBHashKey]
    public string PK { get; set; }    //Partition key

    [DynamoDBRangeKey]
    public string SK { get; set; }    //Sort key

    [DynamoDBGlobalSecondaryIndexRangeKey("Version-index")]
    [DynamoDBProperty]
```

```
public int Version { get; set; }
    . . .
    [DynamoDBProperty]
    public Int64 TTL { get; set; }
}
```

Item access:

```
var _dynamoDbClient = new AmazonDynamoDBClient(AWSCredentials);
var _context = new DynamoDBContext(_dynamoDbClient);

public ProdApp GetProdAppById (Guid id, int version)
{
    var pk = $"{id}-{version}";
    return _context.Load<ProdApp>(pk, ItemType.ProductionApplication);
}
```

For more information, see [Object persistence interface](#) in the DynamoDB documentation.

Document interface

The document interfaces model provides document-based access (similar to XMLDocument in .NET) to a DynamoDB item. This model provides a higher-level programming interface, but translates its calls to low-level APIs to perform the operation.

```
var _dynamoDbClient = new AmazonDynamoDBClient(AWSCredentials);
var _table = Table.LoadTable(_dynamoDbClient, "AppLibrary");

public ProdApp GetProdAppById (Guid id, int version)
{
    var pk = $"{id}-{version}";
    var doc = _table.GetItem(pk, ItemType.ProductionApplication);
    var app = new ProdApp {
        PK = doc["PK"],
        SK = doc["SK"],
        Version = doc["Version"],
        . . .
    };
    return app;
}
```

For more information, see [Document interfaces](#) in the DynamoDB documentation.

Low-level API

The AWS SDK for DynamoDB also provides low-level API access to perform CRUD operations by using the `PutItem`, `GetItem`, `UpdateItem`, and `DeleteItem` methods. This model provides complete control over attribute mapping and type conversions. The response for these calls is a dictionary of key-value pairs.

```
[DynamoDBTable("AppLibrary")]
public class ProdApp
{
    [DynamoDBHashKey]
    public string PK { get; set; }    //Partition key

    [DynamoDBRangeKey]
    public string SK { get; set; }    //Sort key

    [DynamoDBGlobalSecondaryIndexRangeKey("Version-index")]
    [DynamoDBProperty]
    public int Version { get; set; }
    . . .
    [DynamoDBProperty]
    public ProdConfig Config { get; set; }
}

var _dynamoDbClient = new AmazonDynamoDBClient(AWSCredentials);

public ProdApp GetProdAppById (Guid id, int version)
{
    var pk = $"{id}-{version}";
    var resp = _dynamoDbClient.Query(queryRequest);
    var item = resp.Items[0];
    var app = new ProdApp {
        PK = item["PK"].S,
        SK = item["SK"].S,
        Version = Convert.ToInt32(item["Version"].S),
        . . .
        Config = new ProdConfig {
            Name = item["Config"].M["Name"].S,
            Id = Conver.ToInt32(item["Config"].M["Id"].S)
        }
    };
    return app;
}
```

```
}
```

For more information, see [Low-level interfaces](#) in the DynamoDB documentation.

Converters

In some cases, you might have to modify or transform data while saving or reading from the DynamoDB database. In those scenarios, you can use the [IPropertyConverter](#) interface of the [Amazon.DynamoDBv2.DataModel](#) namespace, by using code similar to the following:

```
// Converts the null values of a string property to a valid string and vice versa.
public classNullOrStringConverter : IPropertyConverter
{
    // Called when creating the JSON / DynamoDB item from the model
    public DynamoDBEntry ToEntry(object value)
    {
        var entry = new Primitive
        {
            value = new DynamoDBNull()
        };
        if(value != null)
        {
            entry.Value = value.ToString();
        }
        return entry;
    }
    // Called when populating the model from the JSON / DynamoDB item
    public object FromEntry(DynamoDBEntry entry)
    {
        if(entry is DynamoDBNull)
        {
            return string.Empty;
        }
        else
        {
            return entry.ToString();
        }
    }
}
```

Converter usage in the model:

```
[DynamoDBTable("AppLibrary")]  
public class ProdApp  
{  
    . . .  
  
    [DynamoDBProperty (typeof(NullOrString))]  
    public string AppConfigId { get; set; }  
    . . .  
}
```

Best practices

This section summarizes recommendations in addition to the best practices discussed in previous sections (such as storing items that are greater than 400 KB in Amazon S3, using indexes, single table design, and transactions).

Use the object persistence access pattern

As discussed earlier in this guide, Amazon DynamoDB provides three access patterns. The [object persistence interface](#) provides clean and easy-to-maintain code. Unless you have to customize or transform model property values during read/write operations, we recommend that you use the object persistence interface.

Choose the correct capacity provisioning mode

On-demand capacity provisioning automatically scales read and write operations as your workloads ramp up and down. We recommend that you use this mode if your workload is unpredictable. The cost is usually higher than provisioned capacity mode, and is charged on a pay-as-you-use basis. We recommend that you use provisioned capacity mode when your workload is predictable and you can forecast your capacity requirements. For more information, see the [Pricing models](#) section earlier in this guide.

Use caching

We recommend that you use caching when working with DynamoDB to reduce the cost associated with each read/write operation. Use the appropriate invalidation logic to remove items from the cache when cached data becomes stale. Identify the most frequently used endpoints to implement caching.

Use queries instead of scans

Avoid DynamoDB scans as much as possible. A DynamoDB query is more efficient and less costly than a scan operation. A query filters items based on partition key (PK) and sort key (SK) values, whereas a scan has to read all records to filter items based on the specified parameters. DynamoDB pricing is based on the volume of data read/write operations, so scans incur more cost than queries. Queries are also faster, which ultimately improves the performance of the application.

Validate data integrity

DynamoDB is a NoSQL database, so it doesn't maintain relationship data or include data integrity constraints. It requires each item to have a unique combination of a primary key and sort key only. To ensure data integrity between related items in DynamoDB table, we recommend that you perform strict validations in the application or business layer of your system.

FAQ

This section provides answers to commonly raised questions about using DynamoDB.

What is the maximum table size that I can create in DynamoDB?

There is no limit on the table size or number of columns that you can create.

How many tables can I create per account?

You can create up to 2500 tables for each AWS Region per account. If you want to create more tables, you can request a service quota increase at <https://aws.amazon.com/support>.

How many global secondary indexes can I create on a DynamoDB table?

There is initial quota of 20 global secondary indexes per table. If you want to create more indexes, you can request a service quota increase at <https://aws.amazon.com/support>.

How many items can I add or modify per transaction?

You can add or modify up to 100 items (or 4 MB of data) per transaction. If you want to write more than 100 records to a table, you can use batched write operations.

For a full list of quotas, see [Service, account, and table quotas in Amazon DynamoDB](#) in the DynamoDB documentation.

Next steps and resources

Amazon DynamoDB is designed to be a high-performance NoSQL database. Because of its low cost, high performance, automatic scaling, and other features, it is a good alternative to a relational database system (RDBMS). By following best practices, using indexes, choosing the right partition keys, and carefully designing the table structure, you can get the best out of DynamoDB. You can use DynamoDB Accelerator (DAX) for high-frequency read operations and benefit from fast in-memory performance. If you are looking for an alternative to an RDBMS, consider DynamoDB for its cost and performance benefits.

To get started with DynamoDB, see the following links.

DynamoDB documentation

- [NoSQL design](#)
- [Global Secondary Indexes](#)
- [Transactions](#)
- [Data access – Low-level API](#)
- [Data access – Document model \(mid-level\)](#)
- [Data access – Object persistence model \(high-level\)](#)
- [How it works – Read consistency](#)
- [How it works – Read/write capacity mode](#)
- [Mapping arbitrary data](#)
- [Best practices](#)

AWS Prescriptive Guidance publications

- [Modeling data with Amazon DynamoDB \(guide\)](#)
- [Cross-account full table copy options for Amazon DynamoDB \(guide\)](#)
- [Copy Amazon DynamoDB tables across accounts using a custom implementation \(pattern\)](#)

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an [RSS feed](#).

Change	Description	Date
Updated information	Updated sections on transactional API operations , backup and restore , and the FAQ .	February 24, 2023
Initial publication	—	September 30, 2021

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- **Refactor/re-architect** – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- **Replatform (lift and reshape)** – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- **Repurchase (drop and shop)** – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- **Rehost (lift and shift)** – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- **Relocate (hypervisor-level lift and shift)** – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. This migration scenario is specific to VMware Cloud on AWS, which supports virtual machine (VM) compatibility and workload portability between your on-premises environment and AWS. You can use the VMware Cloud Foundation technologies from your on-premises data centers when you migrate your infrastructure to VMware Cloud on AWS. Example: Relocate the hypervisor hosting your Oracle database to VMware Cloud on AWS.
- **Retain (revisit)** – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later

time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See [attribute-based access control](#).

abstracted services

See [managed services](#).

ACID

See [atomicity, consistency, isolation, durability](#).

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than [active-passive migration](#).

active-passive migration

A database migration method in which in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include SUM and MAX.

AI

See [artificial intelligence](#).

AIOps

See [artificial intelligence operations](#).

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to [the portfolio discovery and analysis process](#) and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see [What is Artificial Intelligence?](#)

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the [operations integration guide](#).

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see [ABAC for AWS](#) in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the [AWS CAF website](#) and the [AWS CAF whitepaper](#).

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

BCP

See [business continuity planning](#).

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see [Data in a behavior graph](#) in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also [endianness](#).

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as "Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see [About branches](#) (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the [Implement break-glass procedures](#) indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and [greenfield](#) strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the [Organized around business capabilities](#) section of the [Running containerized microservices on AWS](#) whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See [AWS Cloud Adoption Framework](#).

CCoE

See [Cloud Center of Excellence](#).

CDC

See [change data capture](#).

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use [AWS Fault Injection Service \(AWS FIS\)](#) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See [continuous integration and continuous delivery](#).

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the [CCoE posts](#) on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to [edge computing](#) technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see [Building your Cloud Operating Model](#).

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post [The Journey Toward Cloud-First & the Stages of Adoption](#) on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the [migration readiness guide](#).

CMDB

See [configuration management database](#).

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or AWS CodeCommit. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision

A field of AI used by machines to identify people, places, and things in images with accuracy at or above human levels. Often built with deep learning models, it automates extraction, analysis, classification, and understanding of useful information from a single image or a sequence of images.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in

an AWS account and Region, or across an organization, by using a YAML template. For more information, see [Conformance packs](#) in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see [Benefits of continuous delivery](#). CD can also stand for *continuous deployment*. For more information, see [Continuous Delivery vs. Continuous Deployment](#).

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see [Data classification](#).

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see [Building a data perimeter on AWS](#).

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See [database definition language](#).

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see [Services that work with AWS Organizations](#) in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See [environment](#).

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see [Detective controls](#) in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally

designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a [star schema](#), a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a [disaster](#). For more information, see [Disaster Recovery of Workloads on AWS: Recovery in the Cloud](#) in the AWS Well-Architected Framework.

DML

See [database manipulation language](#).

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see [Modernizing legacy Microsoft ASP.NET \(ASMX\) web services incrementally by using containers and Amazon API Gateway](#).

DR

See [disaster recovery](#).

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to [detect drift in system resources](#), or you can use AWS Control Tower to [detect changes in your landing zone](#) that might affect compliance with governance requirements.

DVSM

See [development value stream mapping](#).

E

EDA

See [exploratory data analysis](#).

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with [cloud computing](#), edge computing can reduce communication latency and improve response time.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See [service endpoint](#).

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts

or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see [Create an endpoint service](#) in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see [Envelope encryption](#) in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the [program implementation guide](#).

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies,

and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a [star schema](#). It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see [AWS Fault Isolation Boundaries](#).

feature branch

See [branch](#).

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see [Machine learning model interpretability with :AWS](#).

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML

model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

FGAC

See [fine-grained access control](#).

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through [change data capture](#) to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

G

geo blocking

See [geographic restrictions](#).

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see [Restricting the geographic distribution of your content](#) in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the [trunk-based workflow](#) is the modern, preferred approach.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as [brownfield](#). If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries. *Detective guardrails* detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See [high availability](#).

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. [AWS provides AWS SCT](#) that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

laC

See [infrastructure as code](#).

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See [industrial Internet of Things](#).

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently

more consistent, reliable, and predictable than [mutable infrastructure](#). For more information, see the [Deploy using immutable infrastructure](#) best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see [Building an industrial Internet of Things \(IIoT\) digital transformation strategy](#).

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see [What is IoT?](#)

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see [Machine learning model interpretability with AWS](#).

IoT

See [Internet of Things](#).

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the [operations integration guide](#).

ITIL

See [IT information library](#).

ITSM

See [IT service management](#).

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see [Setting up a secure and scalable multi-account AWS environment](#).

large migration

A migration of 300 or more servers.

LBAC

See [label-based access control](#).

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see [Apply least-privilege permissions](#) in the IAM documentation.

lift and shift

See [7 Rs](#).

little-endian system

A system that stores the least significant byte first. See also [endianness](#).

lower environments

See [environment](#).

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see [Machine Learning](#).

main branch

See [branch](#).

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

MAP

See [Migration Acceleration Program](#).

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see [Building mechanisms](#) in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see [Integrating microservices by using AWS serverless services](#).

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see [Implementing microservices on AWS](#).

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial

cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the [AWS migration strategy](#).

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the [discussion of migration factories](#) and the [Cloud Migration Factory guide](#) in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The [MPA tool](#) (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the [migration readiness guide](#). MRA is the first phase of the [AWS migration strategy](#).

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the [7 Rs](#) entry in this glossary and see [Mobilize your organization to accelerate large-scale migrations](#).

ML

See [machine learning](#).

MPA

See [Migration Portfolio Assessment](#).

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see [Strategy for modernizing applications in the AWS Cloud](#).

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see [Evaluating modernization readiness for applications in the AWS Cloud](#).

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see [Decomposing monoliths into microservices](#).

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of [immutable infrastructure](#) as a best practice.

O

OAC

See [origin access control](#).

OAI

See [origin access identity](#).

OCM

See [organizational change management](#).

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See [operations integration](#).

OLA

See [operational-level agreement](#).

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see [Operational Readiness Reviews \(ORR\)](#) in the AWS Well-Architected Framework.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the [operations integration guide](#).

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see [Creating a trail for an organization](#) in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the [OCM guide](#).

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic PUT and DELETE requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also [OAC](#), which provides more granular and enhanced access control.

ORR

See [operational readiness review](#).

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see [Permissions boundaries](#) in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See [personally identifiable information](#).

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

policy

An object that can define permissions (see [identity-based policy](#)), specify access conditions (see [resource-based policy](#)), or define the maximum permissions for all accounts in an organization in AWS Organizations (see [service control policy](#)).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see [Enabling data persistence in microservices](#).

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see [Evaluating migration readiness](#).

predicate

A query condition that returns true or false, commonly located in a WHERE clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see [Preventative controls](#) in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in [Roles terms and concepts](#) in the IAM documentation.

Privacy by Design

An approach in system engineering that takes privacy into account throughout the whole engineering process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see [Working with private hosted zones](#) in the Route 53 documentation.

proactive control

A [security control](#) designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the [Controls reference guide](#) in the AWS Control Tower documentation and see [Proactive controls](#) in *Implementing security controls on AWS*.

production environment

See [environment](#).

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See [responsible, accountable, consulted, informed \(RACI\)](#).

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See [responsible, accountable, consulted, informed \(RACI\)](#).

RCAC

See [row and column access control](#).

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See [7 Rs](#).

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See [7 Rs](#).

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see [Managing AWS Regions](#) in *AWS General Reference*.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See [7 Rs](#).

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See [7 Rs](#).

replatform

See [7 Rs](#).

repurchase

See [7 Rs](#).

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see [Responsive controls](#) in *Implementing security controls on AWS*.

retain

See [7 Rs](#).

retire

See [7 Rs](#).

rotation

The process of periodically updating a [secret](#) to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See [recovery point objective](#).

RTO

See [recovery time objective](#).

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see [About SAML 2.0-based federation](#) in the IAM documentation.

SCP

See [service control policy](#).

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata. The secret value can be binary, a single string, or multiple strings. For more information, see [Secret](#) in the Secrets Manager documentation.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: [preventative](#), [detective](#), [responsive](#), and [proactive](#).

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as [detective](#) or [responsive](#) security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see [Service control policies](#) in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see [AWS service endpoints](#) in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a [service-level indicator](#).

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see [Shared responsibility model](#).

SIEM

See [security information and event management system](#).

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See [service-level agreement](#).

SLI

See [service-level indicator](#).

SLO

See [service-level objective](#).

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid

innovation. For more information, see [Phased approach to modernizing applications in the AWS Cloud](#).

SPOF

See [single point of failure](#).

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a [data warehouse](#) or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was [introduced by Martin Fowler](#) as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see [Modernizing legacy Microsoft ASP.NET \(ASMX\) web services incrementally by using containers and Amazon API Gateway](#).

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use [Amazon CloudWatch Synthetics](#) to create these tests.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see [Tagging your AWS resources](#).

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See [environment](#).

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see [What is a transit gateway](#) in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see [Using AWS Organizations with other AWS services](#) in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the [Quantifying uncertainty in deep learning systems](#) guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See [environment](#).

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see [What is VPC peering](#) in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See [write once, read many](#).

WQF

See [AWS Workload Qualification Framework](#).

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered [immutable](#).

Z

zero-day exploit

An attack, typically malware, that takes advantage of a [zero-day vulnerability](#).

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.